Praktikum Messdatenverarbeitung

Sommersemester 2023

Übung 8: Digitale Filter II

Ongun Türkcüoglu

June 10, 2023

Technische Universität Berlin
Fakultät IV
Institut für Energie und Automatisierungstechnik
Fachgebiet Elektronische Mess- und Diagnosetechnik

Inhaltsverzeichnis

1 Einleitung

1.1 Lernziele

Nach dieser Übungsaufgabe

- wissen Sie, wie FIR-Filter in Fixed-Point-Arithmetik auf einem Mikrocontroller implementiert werden können
- kennen Sie die Abweichungen, die sich durch die Implementierung in Fixed-PointArithmetik ergeben
- wissen Sie, wie man eine Nachabtastung auf einem Mikrocontroller realisieren kann

1.2 Einführung

In der Aufgabe *Digitale Filter II* soll untersucht werden, wie sich das Verhalten von digitalen Filtern bei Implementierung auf stark ressourcenbegrenzten eingebetteten Systemen verändert. Auf diesen können meist nur Ganzzahloperationen (Integer) effizient ausgeführt werden, da sie in der Regel nicht über eine Hardware-Einheit zur Beschleunigung von Gleitkommaoperationen (FPU) verfügen. Eine Implementierung der Gleitkommaoperationen allein in der Software ist zwar möglich, hat aber sehr hohe Laufzeiten zur Folge. Um dennoch Rechnungen mit gebrochen rationalen Zahlen in relativ kurzer Zeit durchführen zu können, bedient man sich in der Regel der **Fixed-Point-Arithmetik**.

Eine Warteschlange (*engl. Queue*) ist besonders gut geeignet, um eine digitale Verzögerungsleitung (*engl. digital delay line, tapped delay line*) zu implementieren, womit eine FIR-Filterung auf dem Mikrocontroller realisiert werden kann.

Eine Nachabtastung **mit** oder **ohne FIR-Filterung** ist über die `ucSerde.daq_config`-Parameter möglich (s. Code 1).

```
from ucSerde import ucSerde as uc

serde = uc('/dev/ttyACM0', 2000000)
serde.daq_config(fclk=72e6, psc=71, arr=49, scount=10000, sfilter=(1 oder 0), sdecim=(1..10))
```

Code 1: Funktionsparameter `ucSerde.daq_config`

2 Vorbereitungsaufgaben

Behalten Sie bei der Bearbeitung der Übungsaufgaben im Hinterkopf, dass es Ihre Abgabe werden soll. Erstellen Sie Abschnitte und strukturieren Sie so die Abgabe. Fügen Sie Fließtext hinzu, indem Sie beschreiben, was Sie gemacht haben und was die Ergebnisse sind. Stellen Sie die Ergebnisse in einer geeigneten Art und Weise dar und interpretieren Sie sie. Achten Sie auch dabei auf Vollständigkeit. Setzen Sie dann den vollständigen Pythoncode gesammelt in den Anhang des Protokolls. Formeln und Berechnungen sollten auch ohne Betrachtung des Pythoncodes bzw. des C-Codes durch das Protokoll verständlich sein.

2.1 Fixed-Point Arithmetik

1. Machen Sie sich mit der Fixed-Point Arithmetik auf einem Mikrocontroller [1] vertraut!

2. Erklären Sie den Aufbau **Q-Zahlenformat**!

3. Erstellen Sie eine Funktion, welche die Reduktion der Genauigkeit durch eine Umwandlung der Filterkoeffizienten vom Gleitkommazahlen-Format in das Q-Zahlenformat zurückgibt. Die Filterkoeffizienten sollen in `Qm.n`-Format mit $m = 0$ und $n = 15$ konvertiert und direkt rückkonvertiert werden. Der Funktionskopf könnte wie folgt aussehen (s. Code 2):

```python
def analyse_qmn(
    xk: np.array, n: int = 15, sel: bool = True
) → (np.array, np.array):
    """Returns the precision error resulting from conversion
    fixed-point ⟷ floating-point

    Parameters
    ----------
    xk : np.array
        array of floating-point or fixed-point numbers
    n : int, optional
        number of bits in fractional part, by default 15
    ret_qpoint : bool, optional
        if False, converts fixed-point to floating-point, if
        True, converts floating-point to fixed-point,
        by default True

    Returns
    -------
    np.array
        result of conversion
    np.array
        precision errors in dB
    """
```

Code 2: Funktionskopf `analyse_qmn`

4. Analysieren Sie den Einfluss des Parameters `N` auf den Frequenzgang des in Übung 7 (*Digitale Filter I*) entwickelten Tiefpass-Filters. Dafür sollen die Koeffizienten des Filters in das Q-Zahlenformat konvertiert werden.

$$Q\text{m.n} \quad \text{mit} \qquad n = 0 \ldots 15, \quad m = 15 - N$$

5. Erstellen Sie eine Python-Funktion, welche Ihre berechneten Filterkoeffizienten in eine C-Header-Datei schreibt. Exportieren Sie die Headerdatei in **include\filter_coeffs.h**. (s. Abs. 2.2, Abb. 1). Die Header-Datei soll wie folgt formattiert sein (s. Code 3):

```
#ifndef FILTER_COEFFS_H
#define FILTER_COEFFS_H

// n: filter order
#define FIR_LENGTH n + 1
#define FIR_COEFFS b0, b1, b2 b3, ... , bn

#endif
```

Code 3: `filter_coeffs.h`

Der Funktionskopf könnte wie folgt aussehen (s. Code 4):

```
def export_fir_coeffs(bn: np.array, fname: str = "filter_coeffs") → None:
    """Export FIR filter coefficients as C-header file

    Parameters
    ----------
    bn : np.array
        Filter coefficients
    fname : str
        Header file name
    """
```

Code 4: Funktionskopf `export_fir_coeffs`

2.2 Mikrocontroller und Digitale-Filter

1. Laden Sie sich das ZIP-Archiv ***mdv-registerlevel-3.zip*** aus dem ISIS-Kurs herunter und extrahieren Sie dies in einem Ordner. Der Ordner soll im Code 1 sein.

2. Ergänzen Sie die TODOs in der Datei **include\filter.h**. Lesen Sie sorgfältig die Dokumentation/Erklärungen in der Headerdatei und implementieren Sie die Funktionen (s. Code 5) so, dass die Anforderungen erfüllt werden.

```
__STATIC_FORCEINLINE bool filter_decim (filter_t *filt);
__STATIC_FORCEINLINE void filter_shift_right (filter_t *filt);
__STATIC_FORCEINLINE void filter_push (filter_t *filt, uint16_t data);
__STATIC_FORCEINLINE bool filter_calc (filter_t *filt);
```

Code 5: `filter`-Funktionen

3. Ergänzen Sie den TODO (s. Code 6) in der Datei **scripts\ucSerde.py** (s. Abb 1).

```
class ucSerde:
   def receive_data(self):
        """Transmits the command to the microcontroller, initializing the
        periphery and starting the analog-digital converter. The converted
        signals are transmitted over UART.
        """
        print("Standby")
        tstart = time.time()
        print("Receiving data")
        self.serialComm.reset_input_buffer()
        data = self.serialComm.read_until(b"/n", size=self.scount * 2)
```

```
        self.serialComm.reset_input_buffer()
        self.serialComm.close()
        tend = time.time()
        print("Time elapsed: {}".format(tend - tstart))

        if self.sfilter == 0:
            it = struct.iter_unpack("<H", data)
            data = np.array(list(it))
        else:
            # TODO
            pass
        return data
```

Code 6: TODO - `ucSerde.receive_data`

4. Erklären Sie dabei ihren Code und beschreiben Sie, wie Sie die Berechnung der Filterantwort umgesetzt haben. Gehen Sie dabei besonders auf die verwendete Fixed-Point-Arithmetik, die Datenstrukturen und die verschiedenen Zahlenformate ein.

5. Erklären Sie den Programmablauf mit einem geeigneten Zustandsautomat[1] oder Flussdiagramm! Beantworten Sie dabei die folgende Fragen:

   (a) Das Programm lässt sich in zwei Phasen unterteilen, welche Phasen erkennen Sie? Wie/wann werden diese Phasen beendet?

   (b) Wie sieht der Programmablauf, wenn es

      i. nicht gefiltert und nicht nachabgetastet,
      ii. gefiltert und nicht nachabgetastet,
      iii. nicht gefiltert und nachabgetastet,
      iv. gefiltert und nachabgetastet wird?

   (c) Wann findet die Nachabtastung statt? Wie ändert sich der Rechenaufwand wenn die FIR-Filterung mit Nachabtastung stattfindet? Begründen Sie Ihre Antwort!

> Analysieren Sie dafür die Datenstruktur `fsm_t` in der Headerdatei include\fsm.h!

[1] Zum Beispiel mit State diagrams | Mermaid

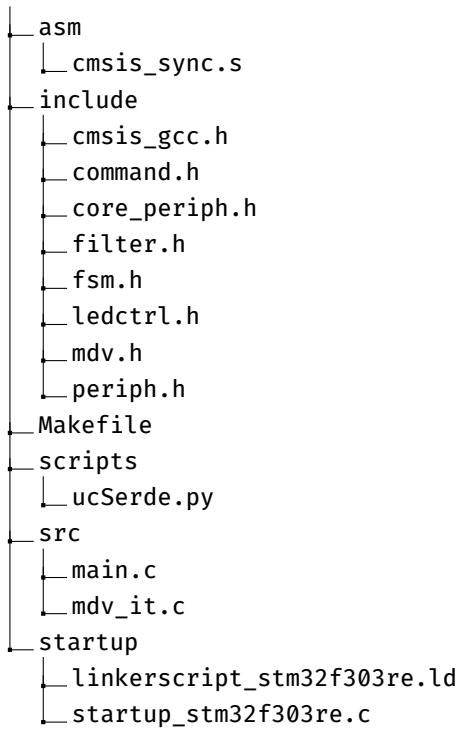

Abbildung 1: Projektstruktur mdv-registerlevel-3

3 Praktikumsaufgaben

Im Labor soll das digitale Tiefpassfilter auf dem bereits in den vorherigen Aufgaben verwendeten Mikrocontroller untersucht werden.

1. **Nehmen Sie den Amplitudenfrequenzgang des im Mikrocontroller implementierten FIR-Filters im Bereich von 0 bis 15 kHz auf. Verwenden Sie eine Abtastrate von 15 kHz.** Vergleichen Sie den Frequenzgang mit dem in Python berechneten.
   (a) Welche Unterschiede fallen Ihnen auf?
   (b) Wie können sie erklärt werden?
2. **Erzeugen Sie mit dem Funktionsgenerator ein verrauschtes 440 Hz-Sinussignal. Nehmen das Signal ohne digitale Filterung einmal mit einer Abtastfrequenz von 3 kHz und einmal mit einer Abtastfrequenz von 15 kHz auf.** Vergessen Sie nicht, das Tiefpassfilter in der Wandler-Box als analoges Anti-Aliasing Filter zu verwenden!
3. **Nehmen sie das gleiche Signal wie in Aufgabe 10 mit einer Abtastfrequenz von 15 kHz, mit anschließender digitaler Filterung und Nachabtastung mit 3 kHz auf!** Vergleichen Sie die Ergebnisse mit denen von Aufgabe 10!
4. Stellen Sie die in den Aufgaben 10 und 11 aufgenommenen Signale sowohl im Zeit- als auch im Frequenzbereich bis zu einer Frequenz von 1,5 kHz dar. Vergleichen Sie die Signale und ihre Spektren! Berechnen Sie für jedes Signal den SNR!
   (a) Welche Unterschiede fallen auf?
   (b) Wie können diese erklärt werden?

Literatur

[1] S. Rein, “Fixed-Point Arithmetic in C: A Tutorial and an Example on Wavelet Filtering,” 2008, https://isis.tu-berlin.de/mod/resource/view.php?id=1610528.